Semantic MediaWiki Model Development through Object-oriented JSON Schema

Simon Heimler, *University of Applied Sciences Augsburg*

***Abstract*—The structure of a Semantic MediaWiki (SMW) can become hard to develop and maintain as it increases in size and complexity. In this paper an object oriented modeling approach based on the JSON Schema format will be introduced. Instead of creating SMW Attributes, Templates and Forms by hand in wikitext, Fields, Models and Forms are defined through JSON Files. Properties can be inherited and overwritten which keeps the model DRY. A Node.js based toolset has been created that validates, visualizes, converts and uploads the model in real-time. While in early stages, this approach worked well for the specific needs of the company that sponsored this project.**

Keywords*—*MediaWiki, Semantic MediaWiki, Semantic Web, JSON Schema.

I. Introduction

In Semantic MediaWikis[11] the data model is defined through semantic attributes and the usage of templates. The Semantic Forms Extension[13] builds on top of that and adds the capability to define forms.

The Attributes, Templates and Forms are implemented as Wiki Pages, using the wikitext syntax. Templates can be reused in different Forms, but almost all the options how to generate the forms are within the forms itself and cannot be shared. Depending on the model this introduces a lot of duplicate code that makes the maintenance and consistency of the model difficult.

There is already a schema based extension available, called Page Schemas[12] which embeds additional XML into the Wiki sites and has a web based GUI. However this extension supports no inheritance. Given the case that two or more forms share the same model but require minor details to be different, the template has to be duplicated and adjusted.

In the following sections an alternative approach based on JSON Schema and Node.js will be introduced.

II. Concept

A. General Concept

The developed toolset is collection of CLI scripts that work as a MediaWiki Bot. Additionally it provides a rudimentary web GUI. The toolset its not integrated into the MediaWiki software in any way, it runs completely independent on the developers computer. This is further explained through the next sections.

B. JSON Schema

JSON Schema[7] is an Internet Engineering Task Force (IETF) standard which is currently in draft state. It is somewhat similar to the better known XML Schema[1] but relies on the simpler JSON format instead of XML, which makes it easier to use and to learn. JSON Schema can be easily extended through custom attributes.

Using a schema in general provides several advantages: With a single schema the user can define the expected structure of the resulting data. Thus it can serve both for client-side and server-side validation.

The schema can not just be used for the obvious validation purposes. It is possible to auto-generate complete forms[6], user documentation[3], random content[9] or UI elements just from the schema information.

C. Excursus: OWL Ontologies

A much more powerful but also more complicated approach to model development using schemas would be OWL. This was even the initial approach to the problem, but there were a few reasons to switch to JSON Schema instead.

The main reason: It is difficult to have hard constraints in an OWL ontology. It is possible to extend OWL to support those, but this would mean to further complicate a complicated system to achieve simple matters.

JSON Schema proved to be sufficient flexible for this project while being much easier to use. Given the case that the more powerful features of ontologies are needed it may be interesting to evaluate the possibility to enhance JSON Schema with ontologies through JSON-LD[10]. That way simple things stay simple with the option for a more complex layer above that foundation.

D. Object oriented approach

To avoid duplicate code in the model it is useful to support features like inheritance. JSON Schema already supports $ref attributes that allow the aggregation of external JSON Schema files. Depending on the implementation this does support inheritance or not.

To avoid misconceptions and make this more obvious the toolset implements its own JSON Schema interpreter which introduces a few additional properties. In this case $extend is introduced. It creates a deep copy of the current object/scope it is in and merges the referenced object in. If an property already exists, the referenced object will overwrite it with its own property. This also implicitly defines the relationships between the different objects. From those a graph oriented database model is generated.

List. 1 is a simple example of an abstract model:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "title": "Shape",
    "description": "Generic Shape",
    "type": "object",

    "properties": {
        "x": {
            "type": "integer"
        },
        "y": {
            "type": "integer"
        }
    },
    "required": ["x", "y"],

    "abstract": true
}
```

Listing 1. /model/_Shape.json

The Circle model (List. 2) will inherit its properties and extend them:

```
{
    "$extend": "/model/_Shape.json",

    "title": "Circle",
    "type": "object",

    "properties": {
        "radius": { "$extend": "/field/radius.json" },
    },
    "required": ["x", "y", "radius"],

    "abstract": false
}
```

Listing 2. /model/Circle.json

Note that the property “radius” is also inherited. The following JSON object (List. 3) just describes a single property. That way properties can be shared between different models.

```
{
    "title": "radius",
    "description": "The radius of a shape",

    "type": "number",
    "minimum": 0,

    "smw_form": {
        "input type": "text with autocomplete"
    }
}
```

Listing 3. /field/radius.json

E. Model structure

To meet the requirements of SMW the model orients itself some along the SMW Structure of Attributes, Templates and Forms. The development model consists of Fields, Models and Forms. While those map roughly to their SMW counterparts, there are differences: Since properties can be inherited from bottom to top, a field can contain information how it should be rendered and validated in the form. Those information are inherited and can be overwritten through the model up to the form (Fig. 1). In SMW Forms this has to be defined in the form itself along with many other information. A SMW attribute contains only the datatype, a template only the names of attributes used and how they should be rendered in the final document.

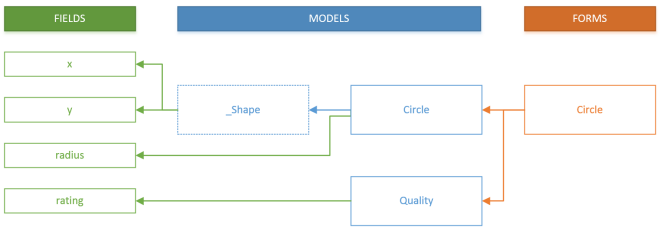

Fig. 1. Model inheritance through Forms, Models and Fields

III. Developed Toolset

A. Technology

The toolset is based on modern web technologies and tools. It is written in JavaScript and uses Node.js[8]. It is a collection of modules and CLI scripts. Automation is provided via the Grunt Task Runner[2]. The user-interface provided through a Node.js webserver as a HTML5 web app.

B. CLI Scripts

The toolset consists of CLI scripts that are run through the Node.js interpreter. Behind those there are various Node.js modules that can be used and shared between the scripts if necessary.

The main script that handles the complete development workflow works as follows:

1) Search the filesystem for JSON Schema files that define the model. Those will be loaded and stored into an internal registry.
2) Search the filesystem for files that contain the last upload state
3) Use a recursive inheritance algorithm which first includes all fields into the models, then handles model to model inheritance and then inherits the models into the forms. While this is being done the toolset also builds a graph structure (that can be imported to Gephi[4] and viewed through the GUI) of the model and validates for common errors.
4) After the model is complete it is converted from JSON Schema to the final wikitext structure. This is done through some preprocessing that handles specific logic and the Handlebars.js[5] template engine. The resulting wikitext with their according sitenames is stored into the internal registry.
5) Calculate the difference between the new state and the last uploaded state if availabe.
6) Use a bot account to log into an external wiki to upload and delete the sites accordingly. A report of the bot activity will be uploaded too.

Several options are available to configure the system, skip specific steps or force others (like full upload).

If the Grunt taskrunner is used it will look for changes in the filesystem and trigger the scripts accordingly in real-time. If the developer changes the content of a field this will automatically trigger the recalculation of the model and the

upload of only those Wiki sites that are affected through this change. See Fig. 2 as an example.

```
Running "shell:generateMarkup" (shell) task

>> FormBot :: generateMarkup
================================================================================
> [WARNING] Model "juristischePerson" is no valid model!
> 151 Properties | 64 Templates | 19 Forms | 56 Categories | 19 Sites
--------------------------------------------------------------------------------
> [WARNING] Form "form:JuristischePerson" has missing model "juristischePerso
n"!
> [WARNING] Edge "template:SoftwareSupport-template:JuristischePerson" is mis
sing its target node "template:JuristischePerson"
> 195 Nodes | 469 Edges
--------------------------------------------------------------------------------
> Generation completed in: 294ms.
--------------------------------------------------------------------------------
 - category:JuristischePerson
 - template:JuristischePerson
 + category:JuristischePersonTYPO
 + template:JuristischePersonTYPO
 C form:JuristischePerson (-1865)
 + Benutzer:FormBot
 + Benutzer:FormBot/2014-09-17_10-45-01
--------------------------------------------------------------------------------
 U (000/007) | Benutzer:FormBot/2014-09-17 10-45-01
 U (001/007) | Kategorie:JuristischePersonTYPO
 U (002/007) | Benutzer:FormBot
 U (003/007) | Formular:JuristischePerson
 U (004/007) | Vorlage:JuristischePersonTYPO
> DELETED: Kategorie:JuristischePerson
> DELETED: Vorlage:JuristischePerson
================================================================================
> Time total: 3457ms.
```

Fig. 2. CLI Script example

C. GUI

To gain some insight about the current state of the model and preview its state in the browser the toolset also provides a web GUI (Fig. 3). Node.js is used as a webserver to provide the sites. The GUI lists and displays the completely calculated and inherited fields, models and forms from the development model. It also lists all generated Wiki sites in the resulting wikitext format.

For a better overview there is an interactive graph view of the model that also visualizes the connections within the model (Fig. 4).

IV. Conclusion

Using JSON Schema for model development has proved to be a simple but sufficient solution. The JSON Schema standard had to be extended to meet some requirements, but it is designed to be easily extensible. More difficult was the adaption of the model to the structure and requirements of Semantic MediaWiki and Semantic Forms. A few compromises had to be made which leads to a slightly less elegant source model. Especially the forms had to implement some rather specific SMW subtleties.

There are also some potential features which could not be implemented: The schema based modeling approach would allow for easy validation on both the front-end and back-end, but neither is supported through SMW right now. Back-end validation however gives a hint if a value doesn't fit the native SMW datatype.

Since there is no easy to use GUI to create and edit the model (right now this is done through a text editor), the user has to familiarize himself with JSON Schema and the additional extensions of it. This would only be rewarding if the model is big enough to regain the time through the automations of the toolset. It would be possible, but very time consuming to create such a GUI.

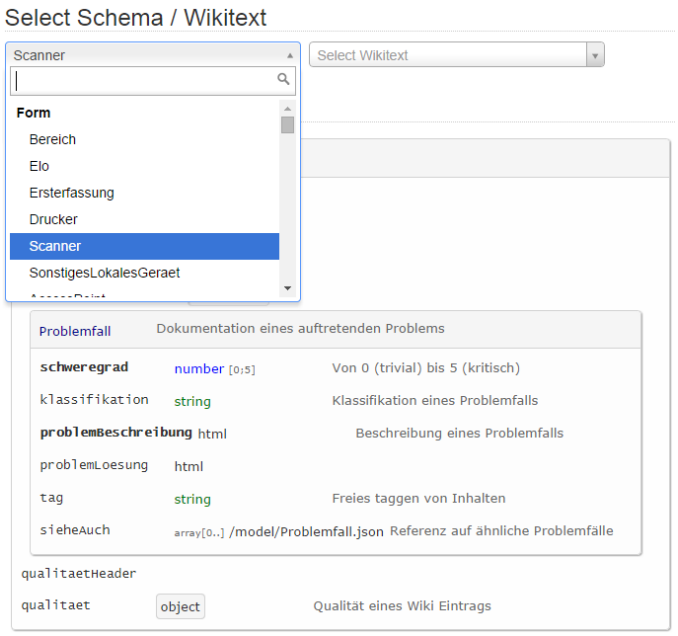

Fig. 3. Web GUI for browsing the calculated model / result

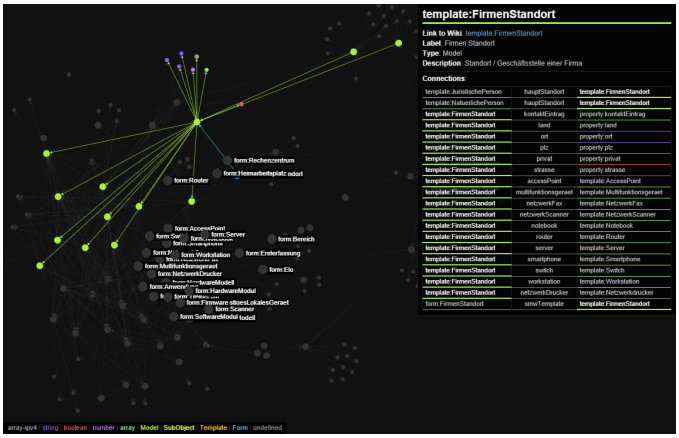

Fig. 4. Interactive Graph View of the model

Summarizing, this approach can be very time saving if the model to develop is more complex. The additional toolset does add to the learning curve however.

Acknowledgment

I would like to thank my advising professor Wolfgang Kowarschick, and the company Computer Bauer for making this research project possible with their support.

REFERENCES

[1] David Beech et al. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C Recommendation. W3C, Apr. 2012. URL: http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.

[2] Bocoup. *Grunt: The JavaScript Task Runner*. URL: http://gruntjs.com/.

[3] Laurent Bovet. *Docson*. URL: https://github.com/lbovet/docson.

[4] Gephi Consortium. *Gephi*. URL: https://gephi.github.io/.

[5] Kevin Decker. *Handlebars.js: Minimal Templating on Steroids*. URL: http://handlebarsjs.com/.

[6] Jeremy Dorn. *JSON Editor*. URL: https://github.com/jdorn/json-editor.

[7] Francis Galiegue, Gary Court. *JSON Schema: core definitions and terminology*. Tech. rep. Internet Engineering Task Force, 2013. URL: http://json-schema.org/latest/json-schema-core.html.

[8] Inc Joyent. *Node.js*. URL: http://nodejs.org/.

[9] Jonah Kagan. *Schematic Ipsum*. URL: http://schematic-ipsum.herokuapp.com/.

[10] Markus Lanthaler, Manu Sporny, and Gregg Kellogg. *JSON-LD 1.0*. W3C Recommendation. http://www.w3.org/TR/2014/REC-json-ld-20140116/. W3C, Jan. 2014.

[11] Semantic MediaWiki Community. *Semantic MediaWiki*. 16.09.2014. URL: https://semantic-mediawiki.org/.

[12] Yaron Koren, Ankit Garg. *Extension:Page Schemas*. 8.05.2014. URL: http://www.mediawiki.org/w/index.php?title=Extension:Page\textunderscoreSchemas\&oldid=999189.

[13] Yaron Koren, Stephan Gambke and others. *Extension:Semantic Forms*. 15.09.2014. URL: http://www.mediawiki.org/w/index.php?title=Extension:Semantic\textunderscoreForms\&oldid=1161499.